# django-freeradius Documentation

*Release 0.1*

**Fiorella De Luca**

**Sep 10, 2020**

# Contents:

Django-freeradius is part of the OpenWISP project.

# CHAPTER 1

# Setup

## 1.1 Create a virtual environment

Please use a python virtual environment. It keeps everybody on the same page, helps reproducing bugs and resolving problems.

We highly suggest to use **virtualenvwrapper**, please refer to the official virtualenvwrapper installation page and come back here when ready to proceed.

```
# create virtualenv
mkvirtualenv radius
```

**Note:** If you encounter an error like `Python could not import the module virtualenvwrapper`, add `VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3` and run `source virtualenvwrapper.sh` again :)

## 1.2 Install required system packages

Install packages required by Weasyprint for your OS:

- Linux
- MacOS
- Windows

## 1.3 Install stable version from pypi

Install from pypi:

```
pip install django-freeradius
```

## 1.4 Install development version

Install tarball:

```
pip install https://github.com/openwisp/django-freeradius/tarball/master
```

Alternatively you can install via pip using git:

```
pip install -e git+git://github.com/openwisp/django-freeradius#egg=django-freeradius
```

If you want to contribute, install your cloned fork:

```
git clone git@github.com:<your_fork>/django-freeradius.git
cd django-freeradius
python setup.py develop
```

## 1.5 Setup (integrate in an existing django project)

In the django `settings.py` file of your project, do the following:

- add `django_freeradius` and `django_filters` to `INSTALLED_APPS`
- set `DJANGO_FREERADIUS_API_TOKEN` (see API Token for more information):

```
INSTALLED_APPS = [
    # other apps
    'django_freeradius',
    'django_filters',
]

DJANGO_FREERADIUS_API_TOKEN = '<a-long-secret-value-of-your-choice>'
```

Add the URLs to your main `urls.py`:

```
urlpatterns = [
    # ... other urls in your project ...

    # django-freeradius urls
    # keep the namespace argument unchanged
    url(r'^', include('django_freeradius.urls', namespace='freeradius')),
]
```

Then run:

```
./manage.py migrate
```

## 1.6 Migrating an existing freeradius database

If you already have a freeradius 3 database with the default schema, you should be able to use it with django-freeradius (and openwisp-radius) easily:

1. first of all, back up your existing database;
2. configure django to connect to your existing database;
3. fake the first migration (which only replicates the default freeradius schema) and then launch the rest of migrations normally, see the examples below to see how to do this.

### 1.6.1 django-freeradius

```
./manage.py migrate --fake django_freeradius 0001_initial_freeradius
./manage.py migrate
```

### 1.6.2 openwisp-radius

In case you are using openwisp-radius:

```
./manage.py migrate --fake openwisp_radius 0001_initial_freeradius
./manage.py migrate
```

## 1.7 Installing for development

Install python3-dev and gcc:

```
sudo apt-get install python3-dev gcc
```

Install sqlite:

```
sudo apt-get install sqlite3 libsqlite3-dev libpq-dev
```

Install mysqlclient:

```
sudo apt-get install libmysqlclient-dev libssl-dev
```

**Note:** If you are on Debian 10 or 9 you may need to install `default-libmysqlclient-dev` instead

Install your forked repo:

```
git clone git://github.com/<your_username>/django-freeradius
cd django-freeradius/
python setup.py develop
```

Install test requirements:

```
pip install -r requirements-test.txt
```

Create database:

```
cd tests/
./manage.py migrate
./manage.py createsuperuser
```

Launch development server:

```
./manage.py runserver
```

You can access the admin interface at http://127.0.0.1:8000/admin/.

Run tests with:

```
./runtests.py
```

## 1.8 Troubleshooting

If you encounter any issue during installation, run:

```
pip install -r requirements.txt -r requirements-test.txt instead of pip install -r␣
↪requirements-test.txt
```

instead of `pip install -r requirements-test.txt`

## 1.9 Automating management commands

Some management commands are necessary to enable certain features and also facilitate database cleanup. In a production environment, it is highly recommended to automate the usage of these commands by using cron jobs.

Edit the crontab with:

```
crontab -e
```

Add and modify the following lines accordingly:

```
# This command deletes RADIUS accounting sessions older than 365 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_radacct␣
↪365

# This command deletes RADIUS post-auth logs older than 365 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_postauth␣
↪365

# This command closes stale RADIUS sessions that have remained open for 15 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py cleanup_stale_
↪radacct 15

# This command deactivates expired user accounts which were created temporarily
# (eg: for en event) and have an expiration date set.
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py deactivate_expired_
↪users

# This command deletes users that have expired (and should have
# been deactivated by deactivate_expired_users) for more than
```

(continues on next page)

(continued from previous page)

```
# 18 months (which is the default duration)
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_users
```

Be sure to replace `<virtualenv_path>` with the absolute path to the Python virtual environment.

Also, change `<full/path/to>` to the directory where `manage.py` is.

To get the absolute path to `manage.py` when django-freeradius is installed for development, navigate to the base directory of the cloned fork. Then, run:

```
cd tests/
pwd
```

More information can be found at the management commands page.

CHAPTER 2

# Available settings

## 2.1 DJANGO_FREERADIUS_EDITABLE_ACCOUNTING

**Default**: `False`

Whether `radacct` entries are editable from the django admin or not.

## 2.2 DJANGO_FREERADIUS_EDITABLE_POSTAUTH

**Default**: `False`

Whether `postauth` logs are editable from the django admin or not.

## 2.3 DJANGO_FREERADIUS_GROUPCHECK_ADMIN

**Default**: `False`

Direct editing of group checks items is disabled by default because these can be edited through inline items in the Radius Group admin (Freeradius > Groups).

*This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.*

If for some reason you need to enable direct editing of group checks you can do so by setting this to `True`.

## 2.4 DJANGO_FREERADIUS_GROUPREPLY_ADMIN

**Default**: `False`

Direct editing of group reply items is disabled by default because these can be edited through inline items in the Radius Group admin (Freeradius > Groups).

*This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.*

If for some reason you need to enable direct editing of group replies you can do so by setting this to `True`.

## 2.5 DJANGO_FREERADIUS_USERGROUP_ADMIN

**Default**: `False`

Direct editing of user group items (`radusergroup`) is disabled by default because these can be edited through inline items in the User admin (Users and Organizations > Users).

*This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.*

If for some reason you need to enable direct editing of user group items you can do so by setting this to `True`.

## 2.6 DJANGO_FREERADIUS_DEFAULT_SECRET_FORMAT

**Default**: `NT-Password`

The default encryption format for storing radius check values.

## 2.7 DJANGO_FREERADIUS_DISABLED_SECRET_FORMATS

**Default**: `[]`

A list of disabled encryption formats, by default all formats are enabled in order to keep backward compatibility with legacy systems.

## 2.8 DJANGO_FREERADIUS_RADCHECK_SECRET_VALIDATORS

**Default**:

```
{'regexp_lowercase': '[a-z]+',
 'regexp_uppercase': '[A-Z]+',
 'regexp_number': '[0-9]+',
 'regexp_special': '[\!\%\-_+=\[\]\
                    {\}\:\,\.\?\<\>\(\)\;]+'}
```

Regular expressions regulating the password validation; by default the following character families are required:

- a lowercase character
- an uppercase character
- a number
- a special character

## 2.9 DJANGO_FREERADIUS_BATCH_DEFAULT_PASSWORD_LENGTH

**Default**: `8`

The default password length of the auto generated passwords while batch addition of users from the csv.

## 2.10 DJANGO_FREERADIUS_BATCH_DELETE_EXPIRED

**Default**: `18`

It is the number of months after which the expired users are deleted.

## 2.11 DJANGO_FREERADIUS_BATCH_PDF_TEMPLATE

It is the template used to generate the pdf when users are being generated using the batch add users feature using the prefix.

The value should be the absolute path to the template of the pdf.

## 2.12 DJANGO_FREERADIUS_API_TOKEN

See API Token.

## 2.13 DJANGO_FREERADIUS_DISPOSABLE_RADIUS_USER_TOKEN

**Default**: `True`

Radius user tokens are used for authorizing users.

When this setting is `True` radius user tokens are deleted right after a successful authorization is performed. This reduces the possibility of attackers reusing the access tokens and posing as other users if they manage to intercept it somehow.

## 2.14 DJANGO_FREERADIUS_API_AUTHORIZE_REJECT

**Default**: `False`

Indicates wether the Authorize API view will return `{"control:Auth-Type":  "Reject"}` or not.

Rejecting an authorization request explicitly will prevent freeradius from attempting to perform authorization with other mechanisms (eg: radius checks, LDAP, etc.).

When set to `False`, if an authorization request fails, the API will respond with `None`, which will allow freeradius to keep attempting to authorize the request with other freeradius modules.

Set this to `True` if you are performing authorization exclusively through the REST API.

## 2.15 `DJANGO_FREERADIUS_API_ACCOUNTING_AUTO_GROUP`

**Default**: `True`

When this setting is enabled, every accounting instance saved from the API will have its `groupname` attribute automatically filled in. The value filled in will be the `groupname` of the `RadiusUserGroup` of the highest priority among the RadiusUserGroups related to the user with the `username` as in the accounting instance. In the event there is no user in the database corresponding to the `username` in the accounting instance, the failure will be logged with *info* level but the accounting will be saved as usual.

## 2.16 `DJANGO_FREERADIUS_EXTRA_NAS_TYPES`

**Default**: `tuple()`

This setting can be used to add custom NAS types that can be used from the admin interface when managing NAS instances.

For example, you want a custom NAS type called `cisco`, you would add the following to your project `settings.py`:

```
DJANGO_FREERADIUS_EXTRA_NAS_TYPES = (
    ('cisco', 'Cisco Router'),
)
```

# CHAPTER 3

# Sending emails to users

Emails can be sent to users whose usernames or passwords have been autogenerated. The content of these emails can be customized with the settings explained below.

## 3.1 DJANGO_FREERADIUS_BATCH_MAIL_SUBJECT

**Default**: `Credentials`

It is the subject of the mail to be sent to the users. Eg: `Login Credentials`.

## 3.2 DJANGO_FREERADIUS_BATCH_MAIL_MESSAGE

**Default**: `username: {}, password: {}`

The message should be a string in the format `Your username is {} and password is {}`.

The text could be anything but should have the format string operator `{}` for `.format` operations to work.

## 3.3 DJANGO_FREERADIUS_BATCH_MAIL_SENDER

**Default**: `settings.DEFAULT_FROM_EMAIL`

It is the sender email which is also to be configured in the SMTP settings. The default sender email is a common setting from the Django core settings under `DEFAULT_FROM_EMAIL`. Currently, `DEFAULT_FROM_EMAIL` is set to to `webmaster@localhost`.

CHAPTER 4

# Installation and configuration of Freeradius 3

This guide explains how to install and configure freeradius 3 in order to make it work with django-freeradius.

**Note:** The guide is written for debian based systems, other linux distributions can work as well but the name of packages and files may be different.

## 4.1 How to install freeradius 3

First of all, become root:

```
sudo -s
```

Let's add the PPA repository for the Freeradius 3.x stable branch:

**Note:** If you use a recent version of Debian like **Stretch** (9) or Ubuntu **Bionic** (18), you should skip the following command and use the official repositories.

```
apt-add-repository ppa:freeradius/stable-3.0
```

Update the list of available packages:

```
apt update
```

These packages are always needed:

```
apt install freeradius freeradius-rest
```

If you use MySQL:

```
apt install freeradius-mysql
```

If you use PostgreSQL:

```
apt install freeradius-postgresql
```

## 4.2 Configuring Freeradius 3

For a complete reference on how to configure freeradius please read the Freeradius wiki, configuration files and their configuration tutorial.

**Note:** The path to freeradius configuration could be different on your system. This article use the `/etc/freeradius/` directory that ships with recent debian distributions and its derivatives

Refer to the mods-available documentation for the available configuration values.

### 4.2.1 Enable the configured modules

First of all enable the `sql`, `rest` and `sqlcounter` modules:

```
ln -s /etc/freeradius/mods-available/sql /etc/freeradius/mods-enabled/sql
ln -s /etc/freeradius/mods-available/rest /etc/freeradius/mods-enabled/rest
ln -s /etc/freeradius/mods-available/sqlcounter /etc/freeradius/mods-enabled/
↪sqlcounter
```

### 4.2.2 Configure the SQL module

Once you have configured properly an SQL server, e.g. PostgreSQL:, and you can connect with a username and password edit the file `/etc/freeradius/mods-available/sql` to configure Freeradius to use the relational database.

Change the configuration for `driver`, `dialect`, `server`, `port`, `login`, `password`, `radius_db` as you need to fit your SQL server configuration.

Refer to the sql module documentation for the available configuration values.

Example configuration using the PostgreSQL database:

```
# /etc/freeradius/mods-available/sql

driver = "rlm_sql_postgresql"
dialect = "postgresql"

# Connection info:
server = "localhost"
port = 5432
login = "<user>"
password = "<password>"
radius_db = "radius"
```

### 4.2.3 Configure the SQL counters

The `sqlcounter` module is used to enforce session limits.

The `mods-available/sqlcounter` should look like the following:

```
# /etc/freeradius/mods-available/sqlcounter

# The dailycounter is included by default in the freeradius conf
sqlcounter dailycounter {
    sql_module_instance = sql
    dialect = ${modules.sql.dialect}

    counter_name = Daily-Session-Time
    check_name = Max-Daily-Session
    reply_name = Session-Timeout

    key = User-Name
    reset = daily

    $INCLUDE ${modconfdir}/sql/counter/${dialect}/${.:instance}.conf
}

# The noresetcounter is included by default in the freeradius conf
sqlcounter noresetcounter {
    sql_module_instance = sql
    dialect = ${modules.sql.dialect}

    counter_name = Max-All-Session-Time
    check_name = Max-All-Session
    key = User-Name
    reset = never

    $INCLUDE ${modconfdir}/sql/counter/${dialect}/${.:instance}.conf
}

# The dailybandwidthcounter is added for django-freeradius
sqlcounter dailybandwidthcounter {
   counter_name = Max-Daily-Session-Traffic
   check_name = Max-Daily-Session-Traffic
   sql_module_instance = sql
   key = 'User-Name'
   reset = daily
   query = "SELECT SUM(acctinputoctets + acctoutputoctets) \
            FROM radacct \
            WHERE UserName='%{${key}}' \
            AND UNIX_TIMESTAMP(acctstarttime) + acctsessiontime > '%%b'"
}
```

**Note:** If your freeradius installation fails to start with an error similar to:

`/etc/raddb/sites-enabled/default[440]:  Failed to find "dailycounter" as a module or policy.`

We need enable the `sqlcounter` in a special way. The `modules` section of `radiusd.conf` should look as shown below. This is because of a bug in freeradius. This should be solved in a future release of freeradius.

```
# /etc/freeradius/radiusd.conf
modules {
    # ..
    $INCLUDE mods-enabled
    $INCLUDE mods-available/sqlcounter
    # ..
}
```

### 4.2.4 Configure the REST module

Configure the rest module by editing the file `/etc/freeradius/mods-enabled/rest`, substituting `<url>` with your django project's URL, (for example, if you are testing a development environment, the URL could be `http://127.0.0.1:8000`, otherwise in production could be something like `https://openwisp2.mydomain.org`)-

Refer to the rest module documentation for the available configuration values.

```
# /etc/freeradius/mods-enabled/rest

connect_uri = "<url>"

authorize {
    uri = "${..connect_uri}/api/v1/authorize/"
    method = 'post'
    body = 'json'
    data = '{"username": "%{User-Name}", "password": "%{User-Password}"}'
    tls = ${..tls}
}

# this section can be left empty
authenticate {}

post-auth {
    uri = "${..connect_uri}/api/v1/postauth/"
    method = 'post'
    body = 'json'
    data = '{"username": "%{User-Name}", "password": "%{User-Password}", "reply": "%
↪{reply:Packet-Type}", "called_station_id": "%{Called-Station-ID}", "calling_station_
↪id": "%{Calling-Station-ID}"}'
    tls = ${..tls}
}

accounting {
    uri = "${..connect_uri}/api/v1/accounting/"
    method = 'post'
    body = 'json'
    data = '{"status_type": "%{Acct-Status-Type}", "session_id": "%{Acct-Session-Id}",
↪ "unique_id": "%{Acct-Unique-Session-Id}", "username": "%{User-Name}", "realm": "%
↪{Realm}", "nas_ip_address": "%{NAS-IP-Address}", "nas_port_id": "%{NAS-Port}", "nas_
↪port_type": "%{NAS-Port-Type}", "session_time": "%{Acct-Session-Time}",
↪"authentication": "%{Acct-Authentic}", "input_octets": "%{Acct-Input-Octets}",
↪"output_octets": "%{Acct-Output-Octets}", "called_station_id": "%{Called-Station-Id}
↪", "calling_station_id": "%{Calling-Station-Id}", "terminate_cause": "%{Acct-
↪Terminate-Cause}", "service_type": "%{Service-Type}", "framed_protocol": "%{Framed-
↪Protocol}", "framed_ip_address": "%{Framed-IP-Address}"}'
    tls = ${..tls}
```

(continues on next page)

(continued from previous page)

```
}
```

### 4.2.5 Configure the site

Configure the `authorize`, `authenticate` and `postauth` section as follows, substituting the occurrences of `<api_token>` with the value of DJANGO_FREERADIUS_API_TOKEN:

```
# /etc/freeradius/sites-enabled/default

server default {

    api_token_header = "Authorization: Bearer <api_token>"

    authorize {
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
        sql
        dailycounter
        noresetcounter
        dailybandwidthcounter
    }

    # this section can be left empty
    authenticate {}

    post-auth {
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest

        Post-Auth-Type REJECT {
            update control { &REST-HTTP-Header += "${....api_token_header}" }
            rest
        }
    }

    accounting {
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
    }
}
```

Please also ensure that `acct_unique` is present in tge `pre-accounting` section:

```
preacct {
    # ...
    acct_unique
    # ...
}
```

### 4.2.6 Restart freeradius to make the configuration effective

Restart freeradius to load the new configuration:

```
service freeradius restart
# alternatively if you are using systemd
systemctl restart freeradius
```

In case of errors you can run freeradius in debug mode by running `freeradius -X` in order to find out the reason of the failure.

**A common problem, especially during development and testing, is that the django-freeradius application may not be running**, in that case you can find out how to run the django development server in the Install for development section.

Also make sure that this server runs on the port specified in `/etc/freeradius/mods-enabled/rest`.

You may also want to take a look at the Freeradius documentation for further information that is freeradius specific.

### 4.2.7 Reconfigure the development environment using PostgreSQL

You'll have to reconfigure the development environment as well before being able to use django-freeradius for managing the freeradius databases.

If you have installed for development, create a file `tests/local_settings.py` and add the following code to configure the database:

```
# django-freeradius/tests/local_settings.py
  DATABASES = {
     'default': {
         'ENGINE': 'django.db.backends.postgresql_psycopg2',
         'NAME': '<db_name>',
         'USER': '<db_user>',
         'PASSWORD': '<db_password>',
         'HOST': '127.0.0.1',
         'PORT': '5432'
     },
  }
```

Make sure the database by the name `<db_name>` is created and also the role `<db_user>` with `<db_password>` as password.

## 4.3 Radius Checks: `is_active` & `valid_until`

Django-Freeradius provides the possibility to extend the freeradius query in order to introduce `is_active` and `valid_until` checks.

An example using MySQL is:

```
# /etc/freeradius/mods-config/sql/main/mysql/queries.conf
authorize_check_query = "SELECT id, username, attribute, value, op \
                         FROM ${authcheck_table} \
                         WHERE username = '%{SQL-User-Name}' \
                         AND is_active = TRUE \
                         AND valid_until >= CURDATE() \
                         ORDER BY id"
```

## 4.4 Using Radius Checks for Authorization Information

Traditionally, when using an SQL backend with Freeradius, user authorization information such as User-Name and "known good" password are stored using the *radcheck* table provided by Freeradius' default SQL schema. Django-Freeradius utilizes Freeradius' rlm_rest module in order to take advantage of the built in user management and authentication capabilities of Django. (See *Configure the REST module* and User authentication in Django)

For existing Freeradius deployments or in cases where it is preferred to utilize Freeradius' *radcheck* table for storing user credentials it is possible to utilize rlm_sql in parallel with (or instead of) rlm_rest for authorization.

**Note:** Bypassing the Django-Freeradius' REST API for authorization means you will have to manually create Radius Check 'password' entries for each user you want to authenticate with Freeradius.

### 4.4.1 Password hashing

By default Django will use PBKDF2 to store all passwords in the database. (See Password management in Django). The default password hashing and storage algorithms in Django are not compatible with those used by Freeradius. Therefore, a default set of Freeradius compatible password storage methods have been provided for deployments that make use of Radius Checks for user credentials.

- Cleartext-Password
- NT-Password
- LM-Password
- MD5-Password
- SMD5-Password
- SHA-Password
- SSHA-Password
- Crypt-Password

**Note:** Only the Crypt-Password hashing attribute is recommended for new entries as it makes use of the sha512_crypt feature supported by most Unix/Linux operating systems. (See passlib.hash) The other password hashing algorithms have been provided for backward compatibility.

### 4.4.2 Configuration

To configure support for accessing user credentials with Radius Checks ensure the `authorize` section of your site as follows contains the `sql` module:

```
# /etc/freeradius/sites-available/default

authorize {
    # ...
    sql  # <-- the sql module
    # ...
}
```

Now you can add new Radius Check entries with one of the supported hashing/storage methods mentioned above.

### 4.4.3 Additional Password Formats

Freeradius supports additional password hashing algorithms which are listed in the Freeradius rlm_pap documentation. If your existing deployment makes use of one of these or you would like to request an addition to Django-Freeradius please see the documentation section on *Contributing*.

Keep in mind that using Radius Checks for accessing user credentials is considered an edge case in Django-Freeradius. Full compatibility with new and existing features is not guaranteed.

## 4.5 Debugging

In this section we will explain how to debug your freeradius instance.

### 4.5.1 Start freeradius in debug mode

When debugging we suggest you to open up a dedicated terminal window to run freeradius in debug mode:

```
# we need to stop the main freeradius process first
service freeradius stop
# alternatively if you are using systemd
systemctl stop freeradius
# launch freeradius in debug mode
freeradius -X
```

### 4.5.2 Testing authentication and authorization

You can do this with `radtest`:

```
# radtest <username> <password> <host> 10 <secret>
radtest admin admin localhost 10 testing123
```

A successful authentication will return similar output:

```
Sent Access-Request Id 215 from 0.0.0.0:34869 to 127.0.0.1:1812 length 75
    User-Name = "admin"
    User-Password = "admin"
    NAS-IP-Address = 127.0.0.1
    NAS-Port = 10
    Message-Authenticator = 0x00
    Cleartext-Password = "admin"
Received Access-Accept Id 215 from 127.0.0.1:1812 to 0.0.0.0:0 length 20
```

While an unsuccessful one will look like the following:

```
Sent Access-Request Id 85 from 0.0.0.0:51665 to 127.0.0.1:1812 length 73
    User-Name = "foo"
    User-Password = "bar"
    NAS-IP-Address = 127.0.0.1
    NAS-Port = 10
    Message-Authenticator = 0x00
    Cleartext-Password = "bar"
Received Access-Reject Id 85 from 127.0.0.1:1812 to 0.0.0.0:0 length 20
(0) -: Expected Access-Accept got Access-Reject
```

Alternatively, you can use `radclient` which allows more complex tests; in the following example we show how to test an authentication request which includes `Called-Station-ID` and `Calling-Station-ID`:

```
user="foo"
pass="bar"
called="00-11-22-33-44-55:localhost"
calling="00:11:22:33:44:55"
request="User-Name=$user,User-Password=$pass,Called-Station-ID=$called,Calling-
↪Station-ID=$calling"
echo $request | radclient localhost auth testing123
```

### 4.5.3 Testing accounting

You can do this with `radclient`, but first of all you will have to create a text file like the following one:

```
# /tmp/accounting.txt

Acct-Session-Id = "35000006"
User-Name = "jim"
NAS-IP-Address = 172.16.64.91
NAS-Port = 1
NAS-Port-Type = Async
Acct-Status-Type = Interim-Update
Acct-Authentic = RADIUS
Service-Type = Login-User
Login-Service = Telnet
Login-IP-Host = 172.16.64.25
Acct-Delay-Time = 0
Acct-Session-Time = 261
Acct-Input-Octets = 9900909
Acct-Output-Octets = 10101010101
Called-Station-Id = 00-27-22-F3-FA-F1:hostname
Calling-Station-Id = 5c:7d:c1:72:a7:3b
```

Then you can call `radclient`:

```
radclient -f /tmp/accounting.txt -x 127.0.0.1 acct testing123
```

You should get the following output:

```
Sent Accounting-Request Id 83 from 0.0.0.0:51698 to 127.0.0.1:1813 length 154
    Acct-Session-Id = "35000006"
    User-Name = "jim"
    NAS-IP-Address = 172.16.64.91
    NAS-Port = 1
    NAS-Port-Type = Async
    Acct-Status-Type = Interim-Update
    Acct-Authentic = RADIUS
    Service-Type = Login-User
    Login-Service = Telnet
    Login-IP-Host = 172.16.64.25
    Acct-Delay-Time = 0
    Acct-Session-Time = 261
    Acct-Input-Octets = 9900909
    Acct-Output-Octets = 1511075509
    Called-Station-Id = "00-27-22-F3-FA-F1:hostname"
```

(continues on next page)

(continued from previous page)

```
    Calling-Station-Id = "5c:7d:c1:72:a7:3b"
Received Accounting-Response Id 83 from 127.0.0.1:1813 to 0.0.0.0:0 length 20
```

## 4.6 Customizing your configuration

You can further customize your freeradius configuration and exploit the many features of freeradius but you will need to test how your configuration plays with *django-freeradius*.

# CHAPTER 5

# Management commands

These management commands are necessary for enabling certain features and for database cleanup.

Example usage:

```
cd tests/
./manage.py <command> <args>
```

In this page we list the management commands currently available in **django-freeradius**.

## 5.1 delete_old_radacct

This command deletes RADIUS accounting sessions older than `<days>`.

```
./manage.py delete_old_radacct <days>
```

For example:

```
./manage.py delete_old_radacct 365
```

## 5.2 delete_old_postauth

This command deletes RADIUS post-auth logs older than `<days>`.

```
./manage.py delete_old_postauth <days>
```

For example:

```
./manage.py delete_old_postauth 365
```

## 5.3 cleanup_stale_radacct

This command closes stale RADIUS sessions that have remained open for the number of specified <days>.

```
./manage.py cleanup_stale_radacct <days>
```

For example:

```
./manage.py cleanup_stale_radacct 15
```

## 5.4 deactivate_expired_users

**Note:** Find out more about this feature in its dedicated page

This command deactivates expired user accounts which were created temporarily (eg: for en event) and have an expiration date set.

```
./manage.py deactivate_expired_users
```

## 5.5 delete_old_users

This command deletes users that have expired (and should have been deactivated by `deactivate_expired_users`) for more than the specified `<duration_in_months>`.

```
./manage.py delete_old_users --older-than-months <duration_in_months>
```

Note that the default duration is set to 18 months.

# CHAPTER 6

# Importing users

This feature can be used for importing users from a csv file. There are many features included in it such as:

- Importing users in batches: all of the users of a particular csv file would be stored in batches and can be retrieved/ deleted easily using the batch functions.
- Set an expiration date: Expiration date can be set for a batch after which the users would not able to authenticate to the RADIUS Server.
- Autogenerate usernames and passwords: The usernames and passwords are automatically generated if they aren't provided in the csv file. Usernames are generated from the email address whereas passwords are generated randomly and their lengths can be customized.
- Passwords are accepted in both cleartext and hash formats from the CSV.
- Send mails to users whose passwords have been generated automatically.

It can be done using both a management command and the admin interface.

## 6.1 batch_add_users

This command imports users from a csv file. Usage is as shown below.

```
./manage.py batch_add_users --name <name_of_batch> \
                            --file <filepath> \
                            --expiration <expiration_date> \
                            --password-length <password_length>
```

Note that the expiration and password-length are optional parameters which default to never and 8 respectively.

## 6.2 Using the admin interface

Selecting the CSV as the strategy and uploading the CSV file is all one will have to do to import the CSV file from the admin interface. It can be checked at */admin/radiusbatch/add*.

It is important to take care of the following when importing users from the CSV.

## 6.3 CSV Format

The CSV shall be of the format:

```
username,password,email,firstname,lastname
```

## 6.4 Imported users with hashed passwords

The hashes are directly stored in the database if they are of the django hash format.

For example, a password myPassword123, hashed using salted SHA1 algorithm, will look like:

```
pbkdf2_sha256$100000$cKdP39chT3pW$2EtVk4Hhm1V65GNfYAA5AHj0uyD60f2CmqumqiB/gRk=
```

So a full CSV line containing that password would be:

```
username,pbkdf2_sha256$100000$cKdP39chT3pW$2EtVk4Hhm1V65GNfYAA5AHj0uyD60f2CmqumqiB/
↪gRk=,email@email.com,firstname,lastname
```

## 6.5 Importing users with clear-text passwords

Clear-text passwords must be flagged with the prefix `cleartext$`.

For example, if we want to use the password `qwerty`, we must use: `cleartext$qwerty`.

## 6.6 Autogeneration of usernames and passwords

Email is the only mandatory field of the CSV file.

Other fields like username and password will be auto-generated if omitted.

### 6.6.1 Batch mail settings

Emails can be sent to users whose usernames or passwords have been autogenerated and contents of these emails can be customized too. Here are some defined settings for doing that:

- DJANGO_FREERADIUS_BATCH_MAIL_SUBJECT
- DJANGO_FREERADIUS_BATCH_MAIL_MESSAGE
- DJANGO_FREERADIUS_BATCH_MAIL_SENDER

# CHAPTER 7

# Generating users

Many a times, a network admin might need to generate temporary users for events etc. This feature can be used for generating users by specifying a prefix and the number of users to be generated. There are many features included in it such as:

- Generating users in batches: all of the users of a particular prefix would be stored in batches and can be retrieved/deleted easily using the batch functions.
- Set an expiration date: Expiration date can be set for a batch after which the users would not able to authenticate to the RADIUS Server.
- PDF: Get the usernames and passwords generated outputted into a PDF.

This can be accomplished from both the admin interface and the management command.

## 7.1 prefix_add_users

This command generates users whose usernames start with a particular prefix. Usage is as shown below.

```
./manage.py prefix_add_users --name <name_of_batch> \
                             --prefix <prefix> \
                             --n <number_of_users>
                             --expiration <expiration_date> \
                             --password-length <password_length>
```

Note that the expiration and password-length are optional parameters which default to never and 8 respectively.

## 7.2 Adding from admin inteface

At the url */admin/django_freeradius/radiusbatch/add* one can directly generate users using the prefix and the number of users. A PDF can be downloaded immediately after the users have been generated.

CHAPTER 8

# Enforcing session limits

The default freeradius schema does not include a table where groups are stored, but django-freeradius adds a model called `RadiusGroup` and alters the default freeradius schema to add some optional foreign-keys from other tables like:

- `radgroupcheck`
- `radgroupreply`
- `radusergroup`

These foreign keys make it easier to automate many synchronization and integrity checks between the `RadiusGroup` table and its related tables but they are not strictly mandatory from the database point of view: their value can be `NULL` and their presence and validation is handled at application level, this makes it easy to use existing freeradius databases.

For each group, checks and replies can be specified directly in the edit page of a Radius Group (`admin` > `groups` > `add group` or `change group`).

## 8.1 Default groups

Some groups are created automatically by **django-freeradius** during the initial migrations:

- `users`: this is the deafult group which limits users sessions to 3 hours and 300 MB (daily)
- `power-users`: this group does not have any check, therefore users who are members of this group won't be limited in any way

You can customize the checks and the replies of these groups, as well as create new groups according to your needs and preferences.

**Note on the default group**: keep in mind that the group flagged as default will by automatically assigned to new users, it cannot be deleted nor it can be flagged as non-default: to set another group as default simply check that group as the deafult one, save and **django-freeradius** will remove the default flag from the old default group.

## 8.2 Freeradius configuration

Ensure the `sqlcounter` module is enabled and configured as described in *Configure the SQL counters*.

# CHAPTER 9

# Registration of new users

Django-freeradius does not ship logic related to registration of new users because there are many good django packages that are aimed at solving that solution.

We recommend using django-rest-auth which provides registration of new users via REST API so you can implement registration and password reset directly from your captive page.

## 9.1 Setup

Install `django-rest-auth` and `django-allauth`:

```
pip install django-rest-auth django-allauth
```

Add the following to your `settings.py`:

```
INSTALLED_APPS = [
    # ... other apps ..
    # apps needed for registration
    'rest_framework.authtoken',
    'rest_auth',
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'rest_auth.registration',
]

SITE_ID = 1
```

Add the rest-auth urls to your main `urls.py`:

```
urlpatterns = [
    # ...
    url(r'^api/v1/rest-auth/', include('rest_auth.urls')),
```

(continues on next page)

(continued from previous page)

```
    url(r'^api/v1/registration/', include('rest_auth.registration.urls'))
]
```

## 9.2 API endpoints

Refer to the django-rest-auth documentation regarding its API endpoints.

## CHAPTER 10

# Registration in openwisp-radius

In openwisp-radius the dependencies and required settings are the same but the additional registration URL route does not need to be added to urls.py because a default route with the built-in registration view is shipped. This is done because the registration needs to take into account multi-tenancy, that is, the system must know which organization the user has to be assigned to when the registration is completed.

In openwisp-radius, the registration URL is:

```
/api/v1/registration/<organization_slug>/
```

CHAPTER 11

# Social Login

Social login is supported by generating an additional temporary token right after users perform the social sign-in, the user is then redirected to the captive page with two querystring parameters: `username` and `token`.

The captive page must recognize these two parameters and automatically perform the submit action of the login form: `username` should obviously used for the username field, while `token` should be used for the password field.

The internal REST API of django-freeradius will recognize the token and authorize the user.

This kind of implementation allows to implement the social login with any captive portal which already supports the RADIUS protocol because it's totally transparent for it, that is, the captive portal doesn't even know the user is signing-in with a social network.

## 11.1 Setup

Install `django-allauth`:

```
pip install django-allauth
```

Ensure your `settings.py` looks like the following (we will show how to configure of the facebook social provider):

```
INSTALLED_APPS = [
    # ... other apps ..
    # apps needed for social login
    'rest_framework.authtoken',
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    # showing facebook as an example
    # to configure social login with other social networks
    # refer to the django-allauth documentation
    'allauth.socialaccount.providers.facebook',
]
```

(continues on next page)

(continued from previous page)

```
SITE_ID = 1

# showing facebook as an example
# to configure social login with other social networks
# refer to the django-allauth documentation
SOCIALACCOUNT_PROVIDERS = {
    'facebook': {
        'METHOD': 'oauth2',
        'SCOPE': ['email', 'public_profile'],
        'AUTH_PARAMS': {'auth_type': 'reauthenticate'},
        'INIT_PARAMS': {'cookie': True},
        'FIELDS': [
            'id',
            'email',
            'name',
            'first_name',
            'last_name',
            'verified',
        ],
        'VERIFIED_EMAIL': True,
    }
}
```

Ensure your main `urls.py` contains the `allauth.urls`:

```
urlpatterns = [
    # .. other urls ...
    url(r'^accounts/', include('allauth.urls')),
]
```

## 11.2 Configure the social account application

Refer to the django-allauth documentation to find out how to complete the configuration of a sample facebook login app.

## 11.3 Captive page button example

Following the previous example configuration with facebook, in your captive page you will need an HTML button similar to the ones in the following examples.

### 11.3.1 django-freeradius

```
<a href="https://openwisp2.mywifiproject.com/accounts/facebook/login/?next=
↪%2Ffreeradius%2Fsocial-login%2F%3Fcp%3Dhttps%3A%2F%2Fcaptivepage.mywifiproject.com
↪%2F%26last%3D"
   class="button">Log in with Facebook
</a>
```

Substitute `openwisp2.mywifiproject.com` and `captivepage.mywifiproject.com` with the hostname of your django-freeradius instance and your captive page respectively.

### 11.3.2 openwisp-radius

This example works for openwisp-radius (multitenant version of django-freeradius), which needs the slug of the organization to assign the new user to the right organization:

```
<a href="https://openwisp2.mywifiproject.com/accounts/facebook/login/?next=
↪%2Ffreeradius%2Fsocial-login%2Fdefault%2F%3Fcp%3Dhttps%3A%2F%2Fcaptivepage.
↪mywifiproject.com%2F%26last%3D"
   class="button">Log in with Facebook
</a>
```

Substitute `openwisp2.mywifiproject.com`, `captivepage.mywifiproject.com` and `default` with the hostname of your openwisp-radius instance, your captive page and the organization slug respectively.

CHAPTER 12

# API Documentation

django-freeradius provides an API that can be used by freeradius to perform the following operations:

- Authorize
- Accounting
- Post Auth

The API also provides other features that can be useful to perform integrations with third-party software:

- Batch User Creation
- Login (Obtain User Auth Token)

## 12.1 API Token

Only requests containing the right API token will able to talk to the API endpoints.

Remember to set API token of your instance by setting `DJANGO_FREERADIUS_API_TOKEN` in your django `settings.py`.

It is highly recommended that you use a hard to guess value, longer than 15 characters containing both letters and numbers. Eg:

```
DJANGO_FREERADIUS_API_TOKEN = "165f9a790787fc38e5cc12c1640db2300648d9a2"
```

HTTP clients must send this token, either in the form of a bearer token or in the form of a query string parameter as shown below.

- Bearer token (recommended):

```
curl -X POST http://localhost:8000/api/v1/authorize/ \
     -H "Authorization: Bearer <token>" \
     -d "username=<username>&password=<password>"
```

- Querystring:

```
curl -X POST http://localhost:8000/api/v1/authorize/?token=<token> \
     -d "username=<username>&password=<password>"
```

Requests which contain an invalid token will receive a `403` HTTP error.

For information on how to configure FreeRADIUS to send the bearer token, see Configure the REST module.

## 12.2 Accounting

```
/api/v1/accounting/
```

### 12.2.1 GET

Returns a list of accounting objects

```
GET /api/v1/accounting/
```

```
[
  {
      "called_station_id": "00-27-22-F3-FA-F1:hostname",
      "nas_port_type": "Async",
      "groupname": null,
      "id": 1,
      "realm": "",
      "terminate_cause": "User_Request",
      "nas_ip_address": "172.16.64.91",
      "authentication": "RADIUS",
      "stop_time": null,
      "nas_port_id": "1",
      "service_type": "Login-User",
      "username": "admin",
      "update_time": null,
      "connection_info_stop": null,
      "start_time": "2018-03-10T14:44:17.234035+01:00",
      "output_octets": 1513075509,
      "calling_station_id": "5c:7d:c1:72:a7:3b",
      "input_octets": 9900909,
      "interval": null,
      "session_time": 261,
      "session_id": "35000006",
      "connection_info_start": null,
      "framed_protocol": "test",
      "framed_ip_address": "127.0.0.1",
      "unique_id": "75058e50"
  }
]
```

### 12.2.2 POST

Add or update accounting information (start, interim-update, stop); does not return any JSON response so that freeradius will avoid processing the response without generating warnings

| Param | Description |
|---|---|
| session_id | Session ID |
| unique_id | Accounting unique ID |
| username | Username |
| groupname | Group name |
| realm | Realm |
| nas_ip_address | NAS IP address |
| nas_port_id | NAS port ID |
| nas_port_type | NAS port type |
| start_time | Start time |
| update_time | Update time |
| stop_time | Stop time |
| interval | Interval |
| session_time | Session Time |
| authentication | Authentication |
| connection_info_start | Connection Info Start |
| connection_info_stop | Connection Info Stop |
| input_octets | Input Octets |
| output_octets | Output Octets |
| called_station_id | Called station ID |
| calling_station_id | Calling station ID |
| terminate_cause | Termination Cause |
| service_type | Service Type |
| framed_protocol | Framed protocol |
| framed_ip_address | framed IP address |

### Pagination

Pagination is provided using a Link header pagination. https://developer.github.com/v3/guides/traversing-with-pagination/

```
{
  ....
  ....
  link: <http://testserver/api/v1/accounting/?page=2&page_size=1>; rel=\"next\",
        <http://testserver/api/v1/accounting/?page=3&page_size=1>; rel=\"last\"
  ....
  ....
}
```

Note: Default page size is 10, which can be overridden using the *page_size* parameter.

### Filters

The JSON objects returned using the GET endpoint can be filtered/queried using specific parameters.

| Filter Parameters | Description |
|---|---|
| username | Username |
| called_station_id | Called Station ID |
| calling_station_id | Calling Station ID |
| start_time | Start time (greater or equal to) |
| stop_time | Stop time (less or equal to) |
| is_open | If stop_time is null |

## 12.3 Authorize

```
/api/v1/authorize/
```

Responds to only **POST**, used for authorizing a given username and password.

```
POST /api/v1/authorize/ HTTP/1.1 username=testuser&password=testpassword
```

| Param | Description |
|---|---|
| username | Username for the given user |
| password | Password for the given user |

See also DJANGO_FREERADIUS_API_AUTHORIZE_REJECT.

## 12.4 PostAuth

```
/api/v1/postauth/
```

Sets the response data to None in order to instruct FreeRADIUS to avoid processing the response body.

Responds only to **POST**.

## 12.5 Batch user creation

```
/api/v1/batch/
```

**Note:** This API endpoint allows to use the features described in *Importing users* and *Generating users*.

Responds only to **POST**, used to save a `RadiusBatch` instance. It returns the information of the batch operation and the list of the users generated. It is possible to generate the users of the `RadiusBatch` with two different strategies: csv or prefix.

The csv method needs the following parameters:

| Param | Description |
|---|---|
| name | Name of the operation |
| strategy | "csv" |
| csvfile | file with the users |
| expiration_date | date of expiration of the users |

These others are for the prefix method:

| Param | Description |
|---|---|
| name | name of the operation |
| strategy | prefix |
| prefix | prefix for the generation of users |
| number_of_users | number of users |
| expiration_date | date of expiration of the users |

## 12.6 Login (Obtain User Auth Token)

```
/api/v1/account/token/
```

**Note:** This endpoint does not require the sending of the *API Token* described in the beginning of this document.

Responds only to **POST**, this endpoint is enabled only if `rest_framework.authtoken` is in `settings.INSTALLED_APPS` (which is optional).

Returns the user access token, which can be used to authenticate the user via the freeradius authorization mechanism.

Parameters:

| Param | Description |
|---|---|
| username | string |
| password | string |

CHAPTER 13

# How to extend django-freeradius

django-freeeadius provieds set of models, admin and API classes which can be imported, extended and hence customized by third party apps.

## 13.1 Update Settings

Update the settings to trigger the swapper:

```
# In settings.py of your project

DJANGO_FREERADIUS_RADIUSREPLY_MODEL = "my_radius_app.RadiusReply"
DJANGO_FREERADIUS_RADIUSGROUPREPLY_MODEL = "my_radius_app.RadiusGroupReply"
DJANGO_FREERADIUS_RADIUSCHECK_MODEL = "my_radius_app.RadiusCheck"
DJANGO_FREERADIUS_RADIUSGROUPCHECK_MODEL = "my_radius_app.RadiusGroupCheck"
DJANGO_FREERADIUS_RADIUSACCOUNTING_MODEL = "my_radius_app.RadiusAccounting"
DJANGO_FREERADIUS_NAS_MODEL = "my_radius_app.Nas"
DJANGO_FREERADIUS_RADIUSUSERGROUP_MODEL = "my_radius_app.RadiusUserGroup"
DJANGO_FREERADIUS_RADIUSPOSTAUTHENTICATION_MODEL = "my_radius_app.RadiusPostAuth"

# where my_radius_app is name of your app extending django_freeradius
```

## 13.2 Extend models

Apart from extending implemented models, django_freeradius also provides flexibility to extend abstract class models from *django-freeradius.base.models*.

Example:

```
# In my_radius_app/models.py

from django.db import models
```

(continues on next page)

(continued from previous page)

```
from django_freeradius.base.models import AbstractRadiusCheck

class RadiusCheck(AbstractRadiusCheck):
    # modify/extend the default behaviour here
    custom_field = models.TextField()
```

Similary, you can extend other model classes from `django_freeradius.base.models`.

## 13.3 Extend admin

Similar to models, abstract admin classes from `django_freeradius.base.admin` can also be extended to avoid duplicate code.

```
# In my_radius_app/admin.py

from django.contrib import admin
from .models import RadiusCheck
from django_freeradius.base.admin import AbstractRadiusAccountingAdmin

class RadiusCheckAdmin(AbstractRadiusCheckAdmin):
    model = RadiusCheck
    # modify/extend default behaviour here
    fields = AbstractRadiusCheckAdmin.fields + ['custom_field']
    list_display = AbstractRadiusCheckAdmin.list_display + ['custom_field']

admin.site.register(RadiusCheck, RadiusCheckAdmin)
```

**Note:** For a real world implementation of extending `django-freeradius.base.admin`, refer openswisp-radius.admin

## 13.4 Extend AppConfig

You can also extend AppConfig class from `django_freeradius.apps.DjangoFreeradiusConfig` and provide support for your signals and hooks.

```
# In my_radius_app/apps.py

from django.conf import settings
from django_freeradius.apps import DjangoFreeradiusConfig
from django.core.exceptions import ImproperlyConfigured

API_TOKEN = settings.DJANGO_FREERADIUS_API_TOKEN

class MyRadiusAppConfig(DjangoFreeradiusConfig):
    name = 'my_radius_app'

    # Overiding DjangoFreeradiusConfig.check_settings
    # just for the sake of example, we add a check which ensures the
    # DJANGO_FREERADIUS_API_TOKEN settings is defined and is at
    # least 20 characters long.
```

(continues on next page)

(continued from previous page)

```
    def check_settings(self):
        if API_TOKEN and len(API_TOKEN) < 20 or not API_TOKEN:
            def check_settings(self):
    if API_TOKEN and len(API_TOKEN) < 20 or not API_TOKEN:
        raise ImproperlyConfigured(
            'Security error: DJANGO_FREERADIUS_API_TOKEN is either not set or is less than 20 characters.')
```

## 13.5 Extend API views

You can also extend API views from `django_freeradius.api.views` to your suit your models.

```
# In my_radius_app/api/views.py

from django_freeradius.api.views import AuthorizeView, AuthorizeView

class RadiusTokenAuthentication(TokenAuthentication):
    # modify/extend default behaviour here
    pass

class RadiusAuthorizeView(AuthorizeView):
    # use your modified authentication class
    authentication_classes = (RadiusTokenAuthentication,)

authorize = RadiusAuthorizeView.as_view()
```

**Note:** For a real world implementation of extending `django-freeradius.api`, refer openwisp-radius.api

# CHAPTER 14

# Contributing

Thank you for taking the time to contribute to django-freeradius.

Follow these guidelines to speed up the process.

**Table of Contents:**

**Note: In order to have your contribution accepted faster**, please read the OpenWISP contributing guidelines and make sure to follow its guidelines.

## 14.1 Setup

Once you have chosen an issue to work on, setup your machine for development

## 14.2 Ensure test coverage does not decrease

First of all, install the test requirements:

```
workon radius  # activate virtualenv
pip install --no-cache-dir -U -r requirements-test.txt
```

When you introduce changes, ensure test coverage is not decreased with:

```
coverage run --source=django_freeradius runtests.py
```

## 14.3 Follow style conventions (PEP8, isort, JSLint)

First of all, install the test requirements:

```
workon radius  # activate virtualenv
pip install --no-cache-dir -U -r requirements-test.txt
npm install -g jslint
```

Before committing your work check that your changes are not breaking the style conventions with:

```
./runflake8
./runisort
jslint ./django_freeradius/static/django-freeradius/js/*.js
```

For more information, please see:

- PEP8: Style Guide for Python Code
- isort: a python utility / library to sort imports

## 14.4 Update the documentation

If you introduce new features or change existing documented behavior, please remember to update the documentation!

The documentation is located in the `/docs` directory of the repository.

To do work on the docs, proceed with the following steps:

```
workon radius  # activate virtualenv
pip install sphinx
cd docs
make html
```

## 14.5 Send pull request

Now is time to push your changes to github and open a pull request!

# CHAPTER 15

# Motivations and Goals

In this page we explain the goals of this project and the motivations that led us on this path.

## 15.1 Motivations

The old version of OpenWISP (which we call OpenWISP 1) had a freeradius module which provided several interesting features:

- user registration
- account verification with several methods
- user management
- password reset
- basic general statistics
- basic user account page with user's statistics

But it also had important problems:

- it was not written with automated testing in mind, so there was a lot of code which the maintainers didn't want to touch because of fear of breaking existing features
- it was not written with an international user-base in mind, it contained a great deal of code which was specific to a single country (Italy)
- it was hard to extend, even small changes required changing its core code
- the user management code was implemented in a different way compared to other openwisp1 modules, which added a lot of maintenance overhead
- it used outdated dependencies which over time became vulnerable and were hard to replace
- **it did not perform hashing of user passwords**
- the documentation did not explain how to properly install and configure the software

Similar problems were affecting other modules of OpenWISP 1, that's why over time we got convinced the best thing was to start fresh using best practices since the start.

## 15.2 Project goals

The main aim of this project is to offer a web application and documentation that helps people from all over the world to implement a wifi network that can use freeradius to authenticate its users, either via captive portal authentication or WPA2 enterprise, **BUT** this doesn't mean we want to lock the software to this use case: we want to keep the software generic enough so it can be useful to implement other use cases that are related to networking connectivity and network management; **just keep in mind our main aim if you plan to contribute to django-freeradius please**.

Other goals are listed below:

- replace the user management system of OpenWISP 1 by providing a similar feature set
- provide a web interface to manage a freeradius database
- provide abstract models and admin classes that can be imported, extended and reused in third party apps
- provide ways to extend the logic of django-freeradius without changing its core
- ensure the code is written with an international audience in mind
- maintain a very good automated test suite
- reuse the django user management logic which is very robust and stable
- ensure passwords are hashed with strong algorithms and freeradius can authorize/authenticate using these hashes (that's why we recommend using the `rml_rest` freeradius module with the REST API of django-freeradius)
- integrate django-freeradius with the rest of the openwisp2 ecosystem
- provide good documentation on how to install the project, configure it with freeradius and use its most important features

# CHAPTER 16

# Indices and tables

- genindex
- modindex
- search